JOURNAL OF THEORETICAL
AND APPLIED MECHANICS
1, 36, 1998

NEW METHOD OF PIVOTING IN THE BLOCK SOLVERS FOR LARGE BANDED LINEAR EQUATION SYSTEMS

MAREK STABROWSKI

Institute of the Theory of Electrical Engineering and Electrical Measurements, Warsaw University of Technology

e-mail: mmst@charlie.iem.pw.edu.pl

New method of pivoting applicable to banded unsymmetric linear equation systems has been introduced. It limits the fill-in and nearly preserves the basic band structure. Two solvers, using a new pivoting method, have been developed. One of these solvers uses elegant indirect addressing and the second relies on explicit shifting of data and explicit pivoting. Both solvers have been written in the C language for two popular UNIX platforms (PC486 and Sun's Sparc5). The details of solvers implementation have been described comprehensively. The performance of both solvers has been analysed theoretically. Quantitative results of the test runs on both platforms have been presented.

Key words: banded linear equation systems, partial pivoting

1. Introduction

This paper deals with solution of large banded unsymmetric linear equation systems without diagonal dominance. Systems of this type appear when solving the problems from various science and technology fields. In particular finite element method (FEM), finite difference method (FDM) and implicit method for ordinary differential equations (ODE) generate such systems. The application areas include electromagnetic field analysis, mechanical stress analysis, kinematics of chemical processes, etc. If no pivoting is necessary, there are available efficient out-of-core solvers for the systems with symmetric (cf George and Liu, 1981; Manoj and Bhattacharyya [4]) or unsymmetric (cf George and Liu, 1981; Manoj and Bhattacharyya [4]; Stabrowski, 1981) matrix of coefficients. These solvers preserve a banded structure of the matrix and the system may be solved "in place". The situation changes dramatically, if the system is unsymmetric, exhibits no diagonal dominance and the pivoting

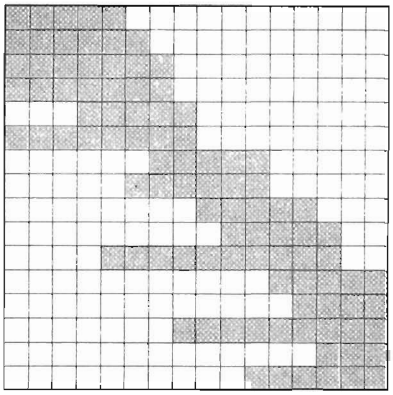

Fig. 1 The profile and bandwidth change as a result of pivoting in a matrix with initial semi-bandwidth equal to 2; pivoted rows: 1←3, 2→4, 3←5, 5←6, 7→9, 9←11, 10←12, 12→14, 13→15, 15→16

is necessary. In this case the pivoting leads to more or less extensive filling of zeros outside of the original non-zero band. Thus the bandwidth grows along with a total number of non-zeros. This process is shown in Fig.1. presenting a small example. The profile (skyline) of the band becomes highly irregular and a total number of non-zeros increases. Most popular approach to similar problems is to use the solver for dense problems (cf Crotty, 1982; Martin and Wikinson, 1971; Stabrowski, 1987). Such solvers ignore the banded structure of the system. In this way the efficiency (in the sense of solution time and disk storage) is severely reduced. Most discouraging is a severe reduction of the system size, resulting from resorting to a dense system solver. Thus some sort of a compromise between the preservation of banded structure and the pivoting with accompanying fill-in should be found.

2. Limited partial pivoting (LPP)

The limited partial pivoting (LPP) is a new original method of partial pivoting preserving the basic banded structure (to some degree) and simplifying overall data storage.

At first let us remind shortly the basics of Crout method used in both solvers described hereinafter. The Crout method has been selected because of the compactness of formulas used. It lends itself very conveniently to enhancement of the stability through introduction of partial double precision in scalar product computation.

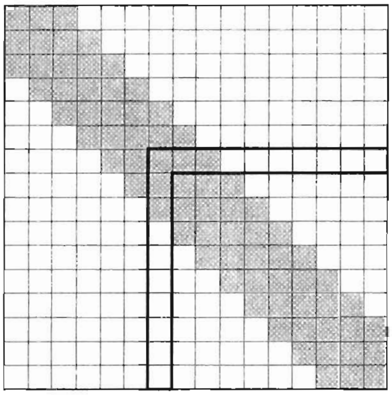

Fig. 2. The active computational front (bold lines) in the Crout method

Forward elimination in the Crout method is performed in the L-shaped active front (Fig.2). The diagonal element and the column elements in the kth step are computed according to the formula

$$l_{ik} = a_{ik} - \sum_{p=1}^{k-1} l_{ip} u_{pk} \qquad i = k, k+1, ..., n \tag{2.1}$$

where

a_{ik} – elements of the extended coefficient matrix

l_{ik} – elements of **L** (lower triangular) matrix

u_{pk} – elements of **U** (upper triangular) matrix.

The elements of the row section in the active front are computed from the formula

$$u_{kj} = \frac{1}{l_{kk}}\Big(a_{kj} - \sum_{p=1}^{k-1} l_{kp} u_{pj}\Big) \qquad j = k+1, ..., n+r \tag{2.2}$$

where

n – number of equations

r – number of right-hand sides.

In the backward substitution the values of solutions s_{k_j} are computed from the formulas

$$s_{kj} = u_{kj} - \sum_{p=n}^{k-1} s_{pj} u_{kp} \qquad j = n+1, ..., n+r \tag{2.3}$$

Partial pivoting, performed during forward elimination, starts with the search for largest (absolute value) coefficient in the current column of the active front. If this largest element, called pivot, is found outside of the current kth row, these rows are interchanged (pivoted). Unrestricted pivoting leads to extensive fill-in already illustrated in Fig.1. The limited partial pivoting limits this fill-in and the expansion of the non-zero band through application of two rather simple rules.

The first rule of LPP restricts the search of pivot to nband rows below the current diagonal element. Here *nband* (the name derived from the C code of the solvers) is the semi-bandwidth of the non-zero band. This limitation is quite natural in the beginning of forward elimination. At this initial stage, there are only zeros beyond *nband* positions below the main diagonal. The first rule of LPP does not limit the growth and expansion of the non-zero band in the course of pivoting.

The second rule of LPP restricts pivoting down (shifting down) of any coefficient row to one such operation. This rule limits, as a consequence, the growth and expansion of the non-zero band to the double value of the original, i.e., to $4nband$. It can be observed (see Fig.1 and Fig.2) that after shifting of a row *nband* positions down, the prediagonal section of this row expands to $2nband$. Conversely the postdiagonal section of a row shifted *nband* positions up expands to $2nband$. Shifting is limited to *nband* by the first rule. This second rule of LPP coupled with the first one limits the band expansion.

The storage scheme of the non-zero band is not connected directly with the LPP strategy. Nevertheless, it can influence the efficiency of a solver using the LPP. Two approaches to the storage have been considered. The first one – modified band scheme introduced originally by Martin and Wikinson (1971) – is rather simple but a bit redundant. The second one – the linked list storage introduced by Jennings (1966) – uses disk and operational memory very economically but the overhead of management seems frustrating. Because of the last feature, Martin's scheme with the total bandwidth expanded to $4nband$ has been selected. The extra width allows one to expand the band to the maximum value following from the LPP strategy.

3. Direct and indirect addressing in lpp block solvers

The limited partial pivoting has been implemented in two solvers written in the C language for two UNIX platforms. The first solver uses indirect addressing in the coefficient matrix (cf Stabrowski, 1966). The second one addresses the coefficients directly and implements all moves explicitly.

The basic idea of indirect addressing may be summed up as elimination of the shifting of large data blocks. The shifting is replaced in some way by modification of the referenced elements addresses.

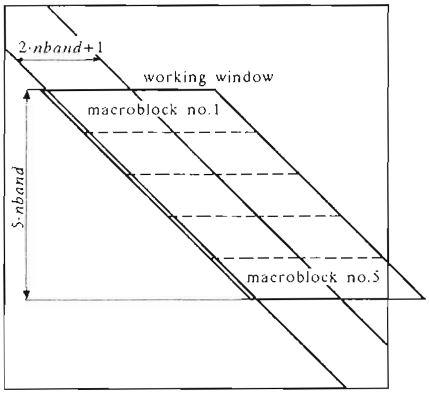

Fig. 3. The working window and its structure in the solver with indirect addressing

The indirect solver nicknamed *ixsolve* (from "indexing solver") looks at the non-zero band through a working window composed of five macroblocks (Fig.3). Every macroblock encompasses *nband* rows, where *nband* is the semi-bandwidth of the equation system. The fourth macroblock is the active one, i.e., the forward elimination is performed in this block. The next – fifth block – is necessary, as the column section of the active front (Fig.2) extends down for *nband* rows. Two macroblocks above the active one are necessary for forming of the scalar products appearing in Eqs (2.1) and (2.2). It should be reminded that the LPP method limits the growth of the non-zero band to $2nband$. One additional block – the first one – must be present in the working window, because of indirect addressing and virtual pivoting. The element located logically in the second macroblock, may be located physically in the first macroblock.

During forward elimination the working window slides down along the non-zero band. However the window is organised as a cyclic buffer with the pointer marking its physical origin. Thus moving a window one macrostep down (i.e. *nband* rows down) means that a new macroblock is read-in from disk memory in the place of the oldest one, i.e., the first. At the same time the pointer of this cyclic buffer is incremented by *nband*, with eventual wrapping at $5nband$. The macroblocks numbers used so far are in fact the logical numbers.

In order to perform pivoting virtually, an information structure has been linked to every row of the coefficient matrix. It is composed of the following members:

- *index* – physical position of a current row
- *diagonal* – physical position of the main diagonal element; strictly speaking it is the shift from initial position
- *before* – number of elements before the main diagonal
- *after* – number of elements after (beyond) the main diagonal.

Interchange of the rows during pivoting is performed through interchange of the *index* members of the structures linked to the corresponding rows. Access to any row starts with its logical number. Next, the *index* value tells where the requested row is located physically. Pivotal interchange should be accompanied by the horizontal shift of interchanged rows. In the solver *ixsolve* this shift is replaced with appropriate modification of the variable *diagonal*. Access to an element of already indirectly addressed row, is made through modification of the logical column number with the value of *diagonal*. The variables *before* and *after* reflect eventual pivoting and fill-in. They are used primarily for limiting the operations with zeros in the scalar products computation.

In the companion solver *noixsolve* (from "nonindexing solver") both shifting operations are performed explicitly. The working window is composed of four macroblocks and its position with respect to the non-zero band is symmetrical. The active macroblock is the macroblock number three. Two preceding blocks are sufficient, as the maximum growth of the non-zero band is limited to $2nband$ and the pivoting is performed explicitly (physically).

If the working window of *noixsolve* slides down one macrostep, the macroblocks number two to four are shifted up in the working window buffer. A new macroblock is read from the disk to the place freed at the bottom of the working window.

During pivotal interchange the rows are interchanged physically with the appropriate horizontal shift. Therefore no information analogous to the variable *diagonal* used in the solver *ixsolve* is necessary. However, the variables *before* and *after* are useful for limiting of the operations in the scalar products computation. The variable *index* is replaced with the logical variable *pivoted*. This variable prevents repeated pivoting of the rows, according to the second rule of the LPP method. It is worth to mention, that in the solver *ixsolve*, the variable *index* has been used for a double purpos. If the value of *index* was different than the logical row number, the pivoting (strictly speaking – pivoting down) of this row was blocked.

4. Analysis of two addressing schemes

Very popular rule for software optimisation recommends avoiding of extensive shifting of the large blocks of data in operational memory. It has been proved for example in the field of sorting algorithms. In the case considered, i.e., in the case of pivoted block solver, at the very beginning this widely accepted rule seemed questionable. Because of this suspicion, after development of the indirect addressing solver, the second one with direct addressing and explicit shifting and pivoting has been developed. Major differences in the sense of computational work may be summed up as follows. The indirect addressing solver computes the indexes (addresses) in the working window in a more complex way and performs the check for wrap-around of the cyclic buffer of working window. On the other hand the direct addressing solver computes the indices of the working windows elements in a simpler way and after finishing with the current macroblock, almost whole contents of the working window is explicitly shifted up.

Let us begin with estimation of the additional computational work in the indirect addressing solver. At first an arbitrary assumption about fill-in will be made setting this value at 25%. The fill-in is defined for current purposes as the additional storage occupied by new non-zeros created during pivoting. The addresses of the band elements are computed basically as the sums of two components – physical row and physical column indexes. Physical row index is computed as the sum of logical row index and the pointer to starting address of the cyclic buffer. Thus one integer addition (symbol t_{iadd} will be used for its execution time) is necessary. Moreover, one check for wrap-around is performed, consuming the time equal to t_{cond} (*cond* – for condition). In the case of physical column computation, only one additional integer sum is computed (variable *diagonal*) contributing t_{iadd} time. The total additional time for single element addressing is equal to

$$t_i = 2t_{iadd} + t_{cond} \tag{4.1}$$

The indirect addressing solver accesses all elements of non-zero band. It follows from the Crout formulas (2.1) and (2.2) that every element (small boundary effect at the end of forward elimination is neglected) is accessed *nband* times.

During the backward substitution only the half of the band is involved into computation – see Eq (2.3). Thus the number of elements accesses for the band expansion coefficient 1.25 may be calculated from the following formula

$$n_{indir} = 1.25 \cdot nband \cdot (3 \cdot nband + 2 \cdot nrhs) \cdot nueq \tag{4.2}$$

where

nband	–	semi-bandwidth
nrhs	–	number of right-hand sides
nueq	–	number of equations.

This formula may be simplified after taking into account the fact that the number of right-hand sides is usually several orders of magnitude lower than the semi-bandwidth

$$n_{indir} = 3.75 \cdot nband^2 \cdot nueq \tag{4.3}$$

Total additional computation time for indirect solver may be estimated after combining of Eqs (4.3) and (4.1)

$$t_{indir} = 3.75 \cdot nband^2 \cdot nueq \cdot (2t_{iadd} + t_{cond}) \tag{4.4}$$

The direct addressing solver uses the logical row and column indexes directly. Therefore, there is no overhead for address computation for this purposes. The overhead arises during the shift of the elements in the working window. The shift of a single element is accompanied by calculation of two addresses (source and destination) and by floating number movement. Every address calculation consumes the time for one integer multiplication t_{imul} and one integer addition t_{iadd}. It follows from quite common procedure of converting the indexes of two-dimensional array into a single index of one-dimensional array or dynamically allocated memory block. Next component of this additional time is the floating number transfer time t_{fmove}. The total time for a single element addressing and move equals

$$t_d = 2(t_{iadd} + t_{imul}) + t_{fmove} \tag{4.5}$$

Every element of the non-zero band is shifted in this way three times inside the working window. It follows from the fact that the working window in the direct addressing solver is composed of four macroblocks. During the backward substitution the shifting is restricted to the right semi-bandwidth and the right-hand sides. After neglecting the share of right-hand sides, total number of shift operations is equal to

$$n_{dir} = 1.25 \cdot 3 \cdot 3 \cdot nband \cdot nueqn = 11.25 \cdot nband \cdot nueq \tag{4.6}$$

After taking into account the time of a single shift operation Eq (4.5), the total additional time for direct addressing solver is equal approximately

$$t_{dir} = 22.5 \cdot nband \cdot nueqn \cdot \left(t_{iadd} + t_{imul} + \frac{1}{2} t_{fmove} \right) \tag{4.7}$$

The ratio of the overhead time for the direct and indirect addressing solvers is

$$\frac{t_{dir}}{t_{indir}} = \frac{6}{nband} \frac{t_{iadd} + t_{imul} + \frac{1}{2} t_{fmove}}{t_{iadd} + t_{cond}} \tag{4.8}$$

It is rather difficult to estimate the exact value of the second quotient. Several simplifying assumptions have been made and the details of specific machine code depend on the compiler. It seems to be safe to assume that this value does not differ significantly from one. The first quotient for real problems is very small, as the semi-bandwidth value *nband* without any doubts is larger than 6. This way, the conclusion that the addressing overhead for direct addressing solver is lower than for the indirect one is proved. It should be noted that the bulk of the processing time – the computation of floating point products - has been left beyond the scope of this analysis.

It is worth at the end of these considerations to point out explicitly the cause of this difference. In indirect addressing solver the additional processing is performed *nband* times for every non-zero coefficient. In direct addressing solver the additional processing (its exact nature and time are different) is carried out only 3 times for every non-zero coefficient.

5. Results of numerical experiments

Both solvers *noixsolve* and *ixsolve* have been developed as C programs for the UNIX platform. Full scalability has been achieved through the usage of dynamic memory allocation for all storage depending on the size of the problem. The solvers have been tested on two popular platforms. The first one – the PC486DX2 – has been used with the USL-Consensys UNIX operating system. The second one – the Sun's Sparc5 – has been used with the Solaris 2.5 operating system. For the test purposes the matrix coefficients have been generated with the aid of random number generator. It has been quite a pleasant surprise to discover that on both platforms the random number series are identical.

The results of test runs for several examples of equation systems are summarised in Table 1 and Table 2. It is very interesting to observe that modest

PC486 has been far better performing than Sparc5. Execution times for PC486 are as low as approximately 55% of the execution times for Sparc5. It is attributable to the excellent floating point performance of the Intel processor and probably better disk services provided by USL UNIX. Both machines have been equipped with comparable SCSI disks. Last, not least also the compilers probably influenced the performance.

Table 1. Typical execution times [s] for example problems run on PC486DX2 (66 MHz) with USL-Consensys UNIX operating system

number of equations	total bandwidth $(2 \cdot nband + 1)$	indirect addressing solver	direct addressing solver
2400	199	76	66
4800	199	146	131
4800	399	767	620
9600	399	1367	1142

Table 2. Typical execution times [s] for example problems run on Sun's Sparc5 (85 MHz) with Solaris 2.5 operating system

number of equations	total bandwidth $(2 \cdot nband + 1)$	indirect addressing solver	direct addressing solver
2400	199	153	126
4800	199	287	230
4800	399	1492	1107
9600	399	2792	2065

Very interesting is the difference between the indirect and direct addressing solvers. The direct addressing solver, according to theoretical analysis, is better than the solver with indirect addressing. On the PC486 platform the difference is of the order of about 20% and on Sparc5 – about 30%. Two important conclusions follow from these observations. First of all, widespread numerical wisdom that extensive shifts and interchanges of the data are very expensive (in the sense of execution time) is far from true in some cases. In the case considered the time for shifting the data has been outweighed by additional effort intensive computing of indirect addresses. Next, it should be mentioned that the indirect addressing solver has been developed as the first one. The direct addressing solver has been developed as the afterthought. It has been hoped that it will prove not only the elegance of indirect addressing but also its efficiency. The result has been exactly the reverse of preliminary expectations. The analysis following the experiments has led to the conclusion that this expectations have been unfounded.

Table 3. The dependence of the execution times [s] on the bandwidth value; example problems run on Sun's Sparc5 (85 MHz) with Solaris 2.5 operating system

number of equations	total bandwidth ($2 \cdot nband + 1$)	indirect addressing solver	direct addressing solver
9600	19	27	26
9600	39	45	44
9600	99	126	111
9600	199	487	419
9600	399	2792	2065

The results presented in Table 3 have been gathered in order to verify more precisely analytical considerations. According to the analysis for equation systems with narrow non-zero band both solvers should come to the solutions in almost the same time. Some traces of such behaviour may be found in this table. For the widest band, indirect addressing solver consumes approximately 30% more time. If the band shrinks to 199, the difference is only 15%. For the total bandwidths of 39 and 19, the execution times are practically equal. However it is a bit risky to trust the data for these extremely narrow bands.

6. Conclusions

New pivoting method nicknamed "limited partial pivoting" has been introduced. It has been implemented in two out-of-core solvers and the results are very satisfying. Both solvers are useful tools solving of large banded unsymmetric systems of linear equations without diagonal dominance. According to analytical considerations the solver with explicit pivoting and direct addressing performs better than its indirect addressing counterpart. However, it is hoped that some reduction of the scope of indirect addressing (e.g. explicit row interchange) may lead to some improvement.

Both solvers have been tested on two different UNIX platforms. The modest PC486DX2 solved the equation systems two times faster than the expensive Sparc5. This spectacular difference is caused by excellent floating point performance of the Intel processor and by lower operating system overhead of USL UNIX.

Introduction of more flexible block structure into the coefficient matrix is envisaged for the future. It is hoped that at the expense of more time-consuming communication with the disk memory and more complex user interface further extension of the application area may be achieved.

References

1. Crotty J.M., 1982, A Block Equation Solver for Large Unsymmetric Matrices Arising in the Boundary Integral Equation Method, *IJNME*, **18**, 997-1017
2. George A., Liu W.H., 1981, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, Englewood Cliffs
3. Jennings A., 1966, A Compact Storage Scheme for the Solution of Symmetric Linear Simultaneous Equations, *Comput. J.*, **9**, 281-285
4. Manoj K.G., Bhattacharyya S.K., A Block Solver for Large Unsymmetric Banded Matrices with Symmetric profiles, *IJNME* (in print)
5. Martin R.S., Wilkinson J.H., 1971, Symmetric Decomposition of Positive Definite Band Matrices, In J.H. Wilkinson, C. Reinsch (edit.), *Handbook for Automatic Computation*, **II**, Springer Verlag, Berlin
6. Rigby R.H., Aliabadi M.H., 19, Out-of-Core Solver for Large, Multi-Zone Boundary Element Matrices, *IJNME*, **38**, 1507-1533
7. Stabrowski M.M., 1981, An Algorithm for Solution of Very Large Banded Unsymmetric Equation Systems, *IJNME*, **16**, 1103-1117
8. Stabrowski M.M., 1987, A Block Equation Solver for Large Unsymmetric Equation Systems with Dense Coefficient Matrices, *IJNME*, **24**, 289-300
9. Stabrowski M.M., 1996, Niektóre problemy rozwiązywania dużych układów sztywnych równań różniczkowych, *Materiały VI Sympozjum "Modelowanie Systemów Pomiarowych*, Krynica

Nowa metoda wyboru elementów głównych w blokowym programie rozwiązywania dużych pasmowych układów równań liniowych

Streszczenie

Przedstawiono nową metodę wyboru elementów głównych dla pasmowych niesymetrycznych układów równań liniowych. Metoda ta ogranicza wypełnienie i zachowuje dość dobrze strukturę pasmową. Opracowano dwa programy stosujące nową metodę. Jeden z nich wykorzystuje elegancki wybór elementów głównych, a drugi stosuje bezpośrednie przemieszczanie danych w pamięci operacyjnej i bezpośredni wybór elementów głównych. Oba programy napisano w języku C dla dwóch popularnych środowisk Unixowych (PC486 i Sparc5 firmy SUN). Obszernie opisano szczegóły realizacji obu programów. Przeprowadzono analizę teoretyczną efektywności obu programów. Przedstawiono rezultaty testów dla obu środowisk sprzętowych.

Manuscript received November 5, 1996; accepted for print May 13, 1997